Walking in a triangulation

Olivier Devillers, Sylvain Pion, Monique Teillaud

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Walking in a triangulation

Olivier Devillers — Sylvain Pion — Monique Teillaud

N° 4120

Fevrier 2001

THÈME 2

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4120--FR+ENG

Walking in a triangulation

Olivier Devillers, Sylvain Pion , Monique Teillaud

Thème 2 — Génie logiciel
et calcul symbolique
Projets Prisme

Rapport de recherche n° 4120 — Fevrier 2001 — Appendix – 4 pages

Abstract: Given a triangulation in the plane or a tetrahedralization in 3-space, we investigate the efficiency of locating a point by walking in the structure with different strategies.

Key-words: Computational Geometry, Delaunay, triangulation, location

This work was partially supported by the ESPRIT IV LTR Project No. 28155 (GALIA).

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 —Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Stratégies de marche dans une triangulation

Résumé : Étant donné une triangulation dans le plan ou une tétrahèdrisation dans l'espace, nous étudions l'efficacité de plusieurs stratégies de localisation par marche.

Mots-clés : Géométrie algorithmique, triangulation, Delaunay, localisation

1 Introduction

Given a triangulation $\mathcal{T}$ of n vertices in the plane and a point p, finding the triangle of $\mathcal{T}$ containing p is a fundamental problem in computational geometry. Several sophisticated structures exist to answer such location queries in optimal $O(\log n)$ time [11, 8] but they are often too complicated and some practitioners may prefer to implement simpler techniques, such as traversing the triangulation using adjacency relations between triangles. This idea can be used directly to locate a point in a triangulation from a known starting point. It is also possible to choose a good starting point in some clever way [10, 9, 4].

There exist different strategies to find the triangle containing the query point p from the triangle containing a source point q. The simplest strategy, that we will call the *straight walk*, consists in visiting all triangles along the line segment qp. A second strategy, *the orthogonal walk*, visits the triangles along an isothetic path moving from q to p by changing one coordinate at a time. Finally, we call *visibility walk* a very popular strategy for the Delaunay triangulation: from a triangle t not containing p we move to the neighbor of t through an edge e if the line supporting e separates t from p; there may be one or two such edges for a triangle t, if there are two we may move to any of these two neighbors. This walk is used for the Delaunay triangulation because in that case it can be proved that it actually reaches the right triangle [6, 3, 7]. In the case of any triangulation the walk may loop. We consider a variant of the visibility walk: the *stochastic walk* in which we decide that if we can choose between two neighbors of t, then the choice is done at random.

All these walking strategies generalize to higher dimensions.

The purpose of this paper is to study the performances of the different strategies from both theoretical and practical points of view, in $\mathbb{R}^2$ and $\mathbb{R}^3$. Hardly anything is known on this topic. The only theoretical result states that the number triangles visited by the straight walk in a Delaunay triangulation of n random points in the plane, to reach a point p from a point q is $O(|qp|\sqrt{n})$, where $|qp|$ denotes the distance from q to p [2, 5].

We are interested in counting not only the number of simplices visited by a walk, but also the cost of visiting one simplex. We consider the robustness issues raised by the implementation of the different strategies.

Section 2 defines the framework of this study. Then we give a detailed description of the different strategies (Sections 3, 4 and 5) in dimensions 2 and 3 together with complexity results. We prove in Section 5.2 that the stochastic walk actually has a zero probability of looping forever, in any dimension. In Section 6 we present some experimental results on the implementation of the different strategies.

2 Framework

Let S be a set of n points in $\mathbb{R}^d, d = 2, 3$. We will consider triangulations (simplicial complexes) whose domain covers the whole convex hull of S. All the simplices of a triangulation are positively oriented.

Given such a triangulation $\mathcal{T}$ of S, we study different strategies to reach a query point p starting from a given starting vertex q of $\mathcal{T}$, walking in $\mathcal{T}$ by using adjacency relations between the simplices of $\mathcal{T}$.

It is not straightforward to decide which strategy is the best one. The paths followed by the different strategies have different lengths in terms of number of simplices. The number of evaluations of *predicates* (simple geometric questions) when visiting a given simplex also depends on the strategy, as well as the nature itself of the predicates involved.

There are theoretical results on the number of triangles visited by the straight walk in the plane, but nothing is known about the visibility walk.

The basic predicate in the straight walk (Section 3) and the visibility walk (Section 5) is the *orientation* predicate, which is defined over $d+1$ points by the sign of a d dimensional determinant, expressed below for 2 and 3 dimensions respectively:

$$\begin{aligned} orientation(\alpha,\beta,\gamma) &= sign\left(\begin{vmatrix} \beta_x-\alpha_x & \gamma_x-\alpha_x \\ \beta_y-\alpha_y & \gamma_y-\alpha_y \end{vmatrix}\right) \\ orientation(\alpha,\beta,\gamma,\delta) &= sign\left(\begin{vmatrix} \beta_x-\alpha_x & \gamma_x-\alpha_x & \delta_x-\alpha_x \\ \beta_y-\alpha_y & \gamma_y-\alpha_y & \delta_y-\alpha_y \\ \beta_z-\alpha_z & \gamma_z-\alpha_z & \delta_z-\alpha_z \end{vmatrix}\right) \end{aligned}$$

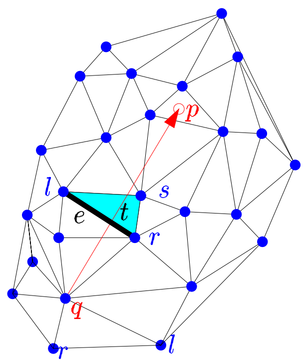
When two points have all but one coordinate equal, the expression of *orientation* simplifies to a determinant of dimension $d-1$. We take advantage of this in the orthogonal walk (Section 4), which uses mostly *comparisons of coordinates* in dimension 2 and more generally *lower dimensional orientation* predicates, which are faster and of course more robust than the full dimensional *orientation* tests, since they involve lower degree computations. The orthogonal walk only uses the d dimensional *orientation* predicate a constant number of times.

The algorithms also use basic operations such as:

- `neighbor(t through pq)` returns the triangle sharing edge `pq` with the triangle `t`.
- `l=vertex of t, l≠q, l≠r;` chooses `l` as the third vertex of a triangle whose two vertices are already known.

(the same notation will be used in the pseudo-code given in Appendix).

These two operations are similar in 3 dimensions for the neighbor of a tetrahedron or the fourth vertex of a tetrahedron. They need a constant number of pointers access or comparisons; the exact number depends on the internal representation of the triangulation, which may be any variant of the DCEL or may be based on simplices or vertices as in CGAL [1].



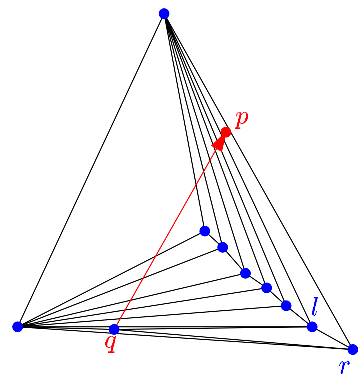


Figure 1: The straight walk.

3 Straight walk

3.1 2 dimensions

This method consists in traversing all the triangles of the triangulation $\mathcal{T}$ that are intersected by the line segment originating from a given vertex q of $\mathcal{T}$ and ending at the query point p. This is performed using the adjacency relations between the triangles.

More precisely, the algorithm first performs an initialization step: from one triangle incident to q we turn around q until a triangle intersected by the ray qp is found. During this initialization step, one orientation test is needed for each visited triangle and the number of visited triangles is at most the degree of q, thus at most n triangles.

Once the initialization step is completed, the straight walk really starts. At a given step of the walk, we traverse some triangle t, and the ray qp goes out of t through edge e. By testing on which side of e lies p, we decide if t contains p or if the walk must go on. In the later case, the walk goes to the neighbor of t through e and the new vertex of that triangle is located with respect to the line qp to decide by which edge of that triangle the ray qp goes out (see Figure 1-left). Therefore, the number of orientation tests performed for each visited triangle is exactly 2. The straight walk cannot visit the same triangle twice, thus the worst case length of a straight walk is at most the number of triangles of the triangulation which is less than $2n$.

Of course, visiting a linear number of triangles seems big, but the general idea of the walking strategy is that in practice you visit less triangles in a reasonable triangulation, although the $2n + O(1)$ bound is tight, as shown by Figure 1-right. In the special case of Delaunay of points evenly distributed, it can be proven that the number of visited triangles during the walk is $O(|pq|\sqrt{n})$ [5].

We give a pseudo-code for a detailed description of the walk (see Appendix).

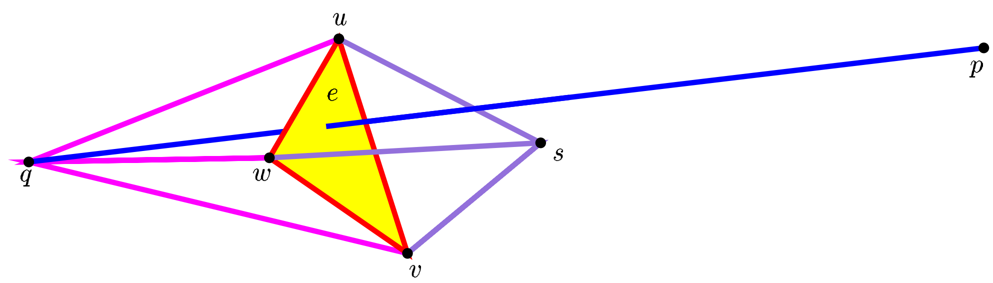


Figure 2: Straight walk in 3 dimensions (main loop).

3.2 3 dimensions

The principle of the walk is similar in higher dimension although a little bit more intricate.

Given vertex q of $\mathcal{T}$ and a query point p, the initialization step consists in finding the tetrahedron incident to q intersected by the ray qp, starting from another tetrahedron incident to q. This problem is in fact the 2 dimensional problem of locating the ray qp in the set of rays having q as origin and triangulated by the tetrahedra incident to q in $\mathcal{T}$. This initialization step is thus solved by the 2D Straight Walk algorithm. Notice that the orientation test for three rays emanating from q is the usual orientation test in three dimensions. The results of the previous paragraph on the number of visited triangles or the number of predicates per triangle apply here.

After this initialization, the main part of the walk begins. At a given step, we know that the ray goes out of some tetrahedron t by a facet e, then we must decide if the walk terminates in t by looking on which side of e lies p (see Figure 2). If the walk continues in the neighbor of t through e, then the ray qp goes out of that neighbor by a facet which is determined by two orientation tests involving q, p, the new vertex and a vertex of e.

Thus the number of orientation tests per visited tetrahedron is exactly 3. As in two dimensions, the number of visited tetrahedra is clearly bounded by the number of tetrahedra of $\mathcal{T}$ since a tetrahedron cannot be visited twice. This number is quadratic in the worst case and a quadratic bound may be reached as shown by the example of Figure 3.

3.3 Degenerate cases

The above algorithms do not handle degenerate cases. When the ray qp goes exactly through a vertex of the triangulation, or through an edge in 3D, the next cell traversed by the ray is not a neighbor of the previous one. In such a case, the algorithm must perform a kind of initialization step to be able to continue the walk.

Actually coding a robust version of the straight walk which handles degenerate cases yields to an intricate code.

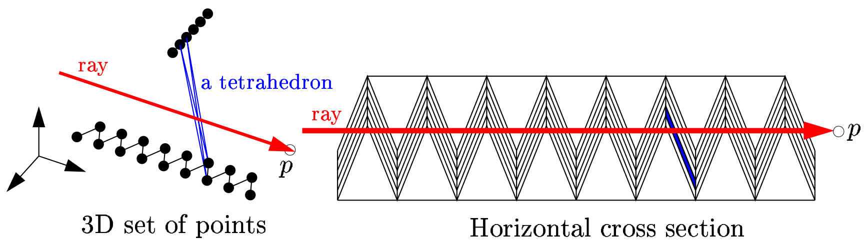


Figure 3: A quadratic example for the straight walk in three dimensions.

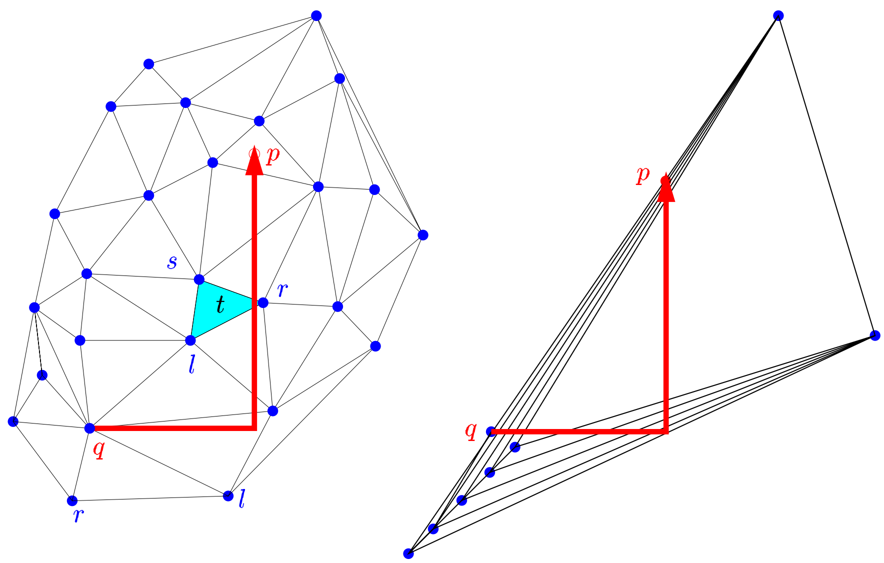


Figure 4: The orthogonal walk.

4 Orthogonal walk

The cost of evaluating an orientation predicate increases with the dimension, thus an idea to improve the efficiency of the algorithm consists in decomposing the walk in pieces parallel to the coordinate axis and to get an *orthogonal* walk (see Figure 4 left).

If the ray pq is parallel to a coordinate axes, then the orientation tests of the straight walk involving both p and q become simpler as noticed in Section 2. This is the case of the orientation tests involved in the initialization phase and of the tests to decide by which edge of the triangle (*resp.* facet of the tetrahedron) the ray goes out. It remains one test per triangle (*resp.* tetrahedron) to decide if the walk ends in that tetrahedron; this test cannot be simplified in general, but a cheaper sufficient condition for the ray to continue can be evaluated first: if p is further than the triangle bounding box in the axis direction, then the walk continues and only otherwise the orientation test is performed.

In the worst case, the orthogonal walk can visit the same simplex at most d times, thus the worst case length of an orthogonal walk is trivially linear in the number of simplices of the triangulation. A bound of $4n + O(1)$ can be reached in 2 dimensions (Figure 4 right). The orthogonal walk can be quadratic in 3 dimensions as shown by the example of Figure 3.

For the special case of the Delaunay triangulation of random points in the plane, the number of visited triangles during the walk is $O((|p\alpha| + |\alpha q|)\sqrt{n})$ [5]. In the orthogonal walk, the dimension of the orientation tests decreases, compared to the straight walk, but the number of visited triangles increases. The average ratio between the length of the straight and the orthogonal walks is the average, on the unit sphere in d dimensions, of the sum of the absolute values of the coordinates, which is d times the average of the absolute value of one coordinate, which we show below to be $4/\pi \approx 1.27$ in 2 dimensions, and $3/2$ in 3 dimensions.

In 2 dimensions :

$$\frac{2}{2\pi}\int_0^{2\pi} |\cos\theta|\, d\theta = \frac{8}{2\pi}\int_0^{\frac{\pi}{2}} \cos\theta\, d\theta = \frac{4}{\pi}\left[\sin\theta\right]_0^{\frac{\pi}{2}} = \frac{4}{\pi}$$

In 3 dimensions :

$$\frac{3\int_0^{\frac{\pi}{2}} \cos\theta\, 2\pi\ \sin\theta\, d\theta}{\int_0^{\frac{\pi}{2}} 2\pi\ \sin\theta\, d\theta} = \frac{3\int_0^{\frac{\pi}{2}} \sin 2\theta\, d\theta}{2} = \frac{3}{2}$$

We give in Appendix a detailed pseudo code description of the algorithm in two dimensions. The two dimensional orientation tests are replaced by comparison of coordinates denoted by `below`, `above`, `left` or `right` in the pseudo code.

5 Visibility and stochastic walks

5.1 Description

The *visibility walk* is extremely simple. Let us describe it in 2D. The 3D case is similar, triangles just have to be replaced by tetrahedra and edges by facets. The algorithm starts from a triangle incident to the starting vertex q. Then, for each visited triangle t, the first edge e is tested. If the line supporting e separates t from p, which reduces to a single orientation test, then the next visited triangle is the neighbor of t through e. Otherwise, the second edge is tested in the same way. In case the test for the second edge also fails, then the third edge is tested. The failure of this third test means that the goal has been reached and that t contains p.

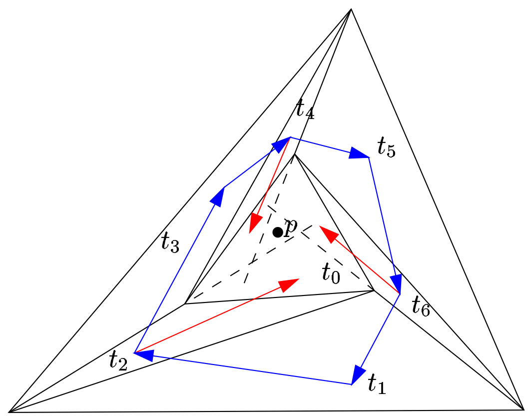


Figure 5: A cycle for the visibility walk.

In addition to its simplicity, the advantage of this walk is that it does not have to deal with degeneracies. If, for an edge e, p lies on the supporting line of e, then the method will look at the next edge. At least one of the edges of each triangle is such that its supporting line strictly separates the triangle from the query point. The only degeneracies to be considered, namely the different cases when p lies on the boundary of a triangle, occur at the end of the walk, when the goal is reached.

The visibility walk is not completely specified: it depends on the implementation of the triangulation, since there is no intrinsic numbering of the edges of a triangle, no intrinsic definition of the “first” edge. The straight walk can be seen as a possible particular execution of the visibility walk algorithm. This is not the case for the orthogonal walk.

The visibility walk in a Delaunay triangulation always terminates, in any dimension [3]. Unfortunately, for non Delaunay triangulations, the visibility walk may fall into a cycle, even in 2D, as illustrated by the famous example of Figure 5. Non-Delaunay triangulations (e.g. the constrained Delaunay triangulation) are also interesting in practice and they cannot be eluded. Therefore, to avoid infinite loops into cycles of non-Delaunay triangulations, a little bit of randomness can be introduced into the algorithm. As already noticed, the visibility walk depends on the numbering of the edges of the triangles. Using this degree of freedom, we may choose between different possible visibility walks.

The *stochastic walk* is obtained by replacing the access to the *first* edge of t by the access to a *random* edge of t. This ensures that, if the walk enters a cycle of the triangulation, it cannot loop into this cycle forever. The termination of the stochastic walk in any kind of triangulation will be proven in the next section.

The stochastic walk performs 1 to 3 orientation tests in each visited triangle. More precisely, suppose a triangle has only one edge whose supporting line separates it from p, then, this edge is chosen as the first one with probability 1/3, and only one test is needed,

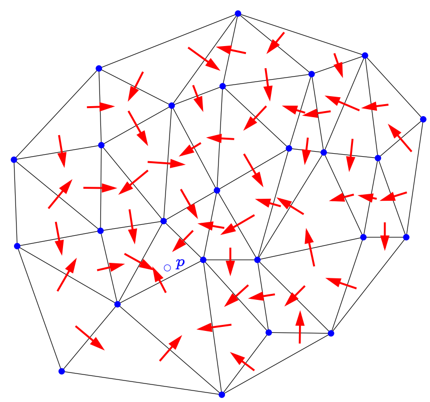


Figure 6: The directed graph $\mathcal{G}$ of neighborhood relationships towards p.

the previous one is chosen with probability 1/3, and two tests are performed, or the next one is chosen, and three tests are performed. This amounts to $1/3 \cdot 1 + 1/3 \cdot 2 + 1/3 \cdot 3 = 2$. In the case when the triangle has two edges whose supporting lines separate it from p, the number of tests is $2/3 \cdot 1 + 1/3 \cdot 2 = 4/3$. Thus, the average number of orientation tests is less than 2, whereas it is 2 for the straight walk. Similar computations show that in 3D, the average number of tests is less than 2.5, whereas it is 3 for the straight walk.

A variant of the stochastic walk is the *remembering stochastic walk* whose pseudo-code is given in Appendix. In a given triangle, the visibility (stochastic or not) walk can test the edge where it comes from, and thus performs an orientation test that was already performed in the previous visited triangle. This can be avoided by remembering, for each visited triangle, the edge that was just crossed by the walk. Then, before testing an edge, it compares it with the remembered edge. This comparison consists of a constant number of comparison of pointers, as mentioned in Section 2. Computations analogous to the ones done above for the variant without memory lead to an average number of orientation tests less than 1.5 in two dimensions and less than 3 in three dimensions. It is not clear whether remembering the edge and performing the comparisons for each triangle is less expensive in practice than a useless orientation test in some triangles. The two variants will be compared experimentally in Section 6.

5.2 Expected validity of the stochastic walk

Let us analyze the algorithm in dimension d.

Given p, we define the directed graph $\mathcal{G}$, from $\mathcal{T}$, as follows. The nodes of $\mathcal{G}$ are the simplices of $\mathcal{T}$ (we will use the same notation for a simplex and its associated node), and there is an oriented arc from node t to node t' if the corresponding simplices are adjacent through a facet e, in such a way that t' and p lie on the same side of e (see Figure 6).

Lemma 1 *Given a facet e shared by simplices t and t' and such that the arc of $\mathcal{G}$ is oriented from node t to node t', the probability that a stochastic path reaching t goes to t' is greater than $\frac{1}{d+1}$.*

Proof: If the path goes through t, then a facet of t having p on the other side must be chosen to continue the stochastic path. e may be chosen first, then the stochastic path uses it since p is on the good side of e; this happens with probability $\frac{1}{d+1}$ (the $d+1$ facets have equal probability). If another facet is chosen first, then p may be on the wrong side and e can be chosen after; this happens with a probability depending of the geometric configuration, the probability is just lower bounded by zero to get the result of the lemma. ■

Theorem 2 *Given a triangulation $\mathcal{T}$ and a query point p in dimension d, the stochastic walk terminates with probability 1.*

Proof: The out-degree of a node of $\mathcal{G}$ is between 1 and d. As noticed before, the graph $\mathcal{G}$ may have cycles, but we will prove that the stochastic walk cannot cycle forever and will necessarily reach the only sink of the graph $\mathcal{G}$, i.e. the simplex S containing p.

Let us label all the nodes of $\mathcal{G}$ by their distances to S in $\mathcal{G}$, where the distance between a node to S is the minimum number of arcs to be followed to reach S from this node (by the definition of $\mathcal{G}$, there is always a path from any node to S). Then by construction, for any node t of label k, there exists an arc of $\mathcal{G}$ from t to at least one node of label $k-1$. Thus if t is visited, then a node of label $k-1$ is visited with probability higher than $\frac{1}{d+1}$ by Lemma 1.

Assume that a stochastic walk visits N_k nodes of label k. Then $N_{k-1} \geq \frac{N_k}{d+1}$ and by an immediate induction: $N_k \leq (d+1)^k N_0$. This relation clearly proves that $N_0 \neq 0$. So, the walk terminates and reaches S, which is the only node of label 0. ■

Additionally, this proof yields an exponential bound on the length of the stochastic walk:

$$Cost \leq \sum_{k=0}^{\Delta} N_k \leq \sum_{k=0}^{\Delta} (d+1)^k \leq \frac{(d+1)^{\Delta+1}}{d}$$

($N_0 = 1$) where Δ is the maximal length (in terms of number of arcs) of a shortest path in $\mathcal{G}$ from any node to S. Since the straight walk is a particular case of visibility walk, Δ is bounded by the longest straight walk in the triangulation, that is $\Delta = O(n)$ in two dimensions and $\Delta = O(n^2)$ in three dimensions.

Unfortunately, for very special configurations of points, this exponential length of the stochastic walk can actually be reached.

The triangulation depicted on Figure 7 consists of one central triangle containing the point p to be located and k layers of cycles around it. These cycles go through a rectangle formed by $k \times k^2$ small squares.

Any triangle having two outgoing arcs in graph $\mathcal{G}$, in this example, is as shown in Figure 7: the ingoing edge e is chosen first with probability $1/3$, then the walk must cross the next

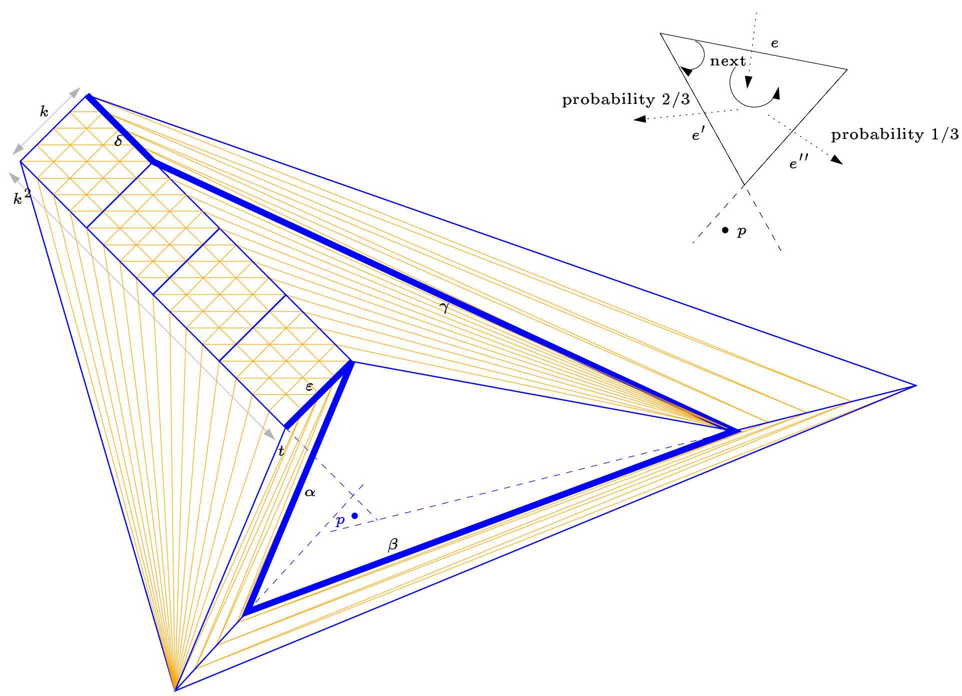


Figure 7: The stochastic walk may have exponential length.

edge e', which forces the walk to follow a cycle. e' can also be chosen first with probability 1/3. So, the walk stays in the cycle with probability 2/3. The edge e'' allows the walk to leave the cycle. It is crossed only when it is chosen first, which occurs with probability 1/3.

Let a stochastic walk start in triangle t defined in the figure.
It reaches p through edge α if, for each visited triangle, it chooses the edge out of the cycle, which occurs with probability $(1/3)^k$. It reaches p through edge β if it chooses i times the edge of the cycle, then it chooses the edge in the cycle, then it chooses $k-i$ times the edge out of the cycle. This occurs with probability $k.(2/3).(1/3)^k$. Analogously, it crosses edge γ with probability $\binom{k+1}{2}.(2/3)^2.(1/3)^k$.
Thus, the walk enters the $k \times k^2$ rectangle by one of the k edges δ with probability $1-(1/3)^k - k.(2/3).(1/3)^k - \binom{k+1}{2}.(2/3)^2.(1/3)^k$.
Let us consider a path entering the rectangle through δ. Giving a tight bound on the probability that such a path goes out of the rectangle through one edge of ε is quite complicated. Let us use a loose bound equal to $(1/2)^{k-1}$. If the path does not go out through ε, then it necessarily reaches its starting triangle t, and using the previous bound, this occurs with probability greater than $1-(1/2)^{k-1}$.
Summarizing, a path starting at t reaches t again with probability greater than

$$\left(1-(1/3)^k - k.(2/3).(1/3)^k - \tbinom{k+1}{2}.(2/3)^2.(1/3)^k\right)\left(1-(1/2)^{k-1}\right) \geq 1 - \frac{2k^2+8k+9}{3^{k+2}} - \frac{1}{2^{k-1}}.$$

So, the expected number of times that t is visited is greater than $\sum_{j=0}^{\infty}\left(1-\frac{2k^2+8k+9}{3^{k+2}}-\frac{1}{2^{k-1}}\right)^j = \frac{1}{\frac{2k^2+8k+9}{3^{k+2}}+\frac{1}{2^{k-1}}}$. The shortest cycle from t to t is obtained by traversing the triangles having a vertex on the convex hull of the points. Its length is k^2+2k+6. So, we get an expected length for stochastic walk greater than $\frac{k^2+2k+6}{\frac{2k^2+8k+9}{3^{k+2}}+\frac{1}{2^{k-1}}} \geq 2^{k+1}$.
Moreover, this triangulation has $(k+1)(k^2+1)+2(k+1) \leq (k+1)^3$ vertices. Thus, we proved the following result:

Theorem 3 *There exists a triangulation $\mathcal{T}$ of n points in dimension 2 and a query point p, such that the expected length of the stochastic walk is greater than $2^{\sqrt[3]{n}}$.*

6 Experimental results

We experimented different walking strategies for locating points in a Delaunay triangulation of 100.000 or 1.000.000 points evenly distributed in a square or in a cube. The walk was performed in the standard way, starting the walk at some known vertex of the triangulation, or as a tool in the Delaunay hierarchy [4] which walks in a hierarchy of more and more refined samples; using this method, locating a query involves few ($O(\log n)$) walks visiting a relatively small number of triangles.

The algorithms are coded in C++. The orientation tests use the usual floating point arithmetic. Robustness issues due to degenerate cases or rounded computations are solved by perturbation and static filtering, which do not have a significant influence on the running time for these random data.

For each strategy we count the number of visited triangles or tetrahedra ($\sharp\Delta$), the number of full dimensional orientation predicates ($\sharp$orient) and the running time (benchmarks are done on a Sun Ultra10 440MHz, 768Mo; times are obtained with the `clock` command, and averaged on 500.000 locations in 5 different random triangulations, times for each run are given in appendix).

The four strategies presented above are compared in the following tables. In fact, since the tests were performed on Delaunay triangulations, the visibility walk (without randomness) does not cycle and produces results similar to the stochastic walk, in both basic and remembering versions.

	100.000 points						1.000.000 points					
	Walk			Hierarchy			Walk			Hierarchy		
	$\sharp\Delta$	$\sharp$orient	μs	$\sharp\Delta$	$\sharp$orient	μs	$\sharp\Delta$	$\sharp$orient	μs	$\sharp\Delta$	$\sharp$orient	μs
	per point			per point			per point			per point		
Stochastic 2D	74	133	65	23	49	41	225	391	161	29	61	48
Visibility 2D	78	136	64	23	44	39	224	386	152	29	56	47
Rem. stoch. 2D	70	126	65	23	49	42	220	382	168	29	61	49
Rem. visib. 2D	77	104	62	23	35	39	224	296	147	28	42	45
Straight 2D	72	145	68	20	42	43	193	387	158	24	52	50
Orthogonal 2D	84	6	62	27	9	43	283	5	162	33	11	50
Visibility 3D	184	381	187	30	74	65	359	736	420	36	87	81
Stochastic 3D	167	325	183	30	69	65	335	641	402	35	80	82
Rem. visib. 3D	184	301	178	30	59	66	359	581	403	35	67	81
Rem. stoch. 3D	167	261	176	30	56	64	335	516	392	35	65	81
Straight 3D	157	466	204	25	75	70	309	919	438	31	91	86
Orthogonal 3D	198	11	206	42	21	86	417	12	452	48	21	109

The running times of all strategies are of the same order.

The straight walk has the best performances in terms of visited simplices, both theoretically and experimentally, but it has the worst cost per triangle. Another drawback of the straight walk is the management of degenerate cases which make the code quite intricate, especially in three dimensions.

For walks of large length in terms of visited simplices, the orthogonal walk is faster. In fact it will be the right choice when using expensive arithmetic (e.g. multi-precision exact arithmetic).

7 Conclusion and open problems

We presented four strategies for walking in a triangulation to locate a point: the straight walk, the visibility walk with or without memory, and the orthogonal walk. We studied them from both theoretical and practical points of view.

The best method to implement is the stochastic visibility walk, since it performs experimentally a little bit better than the straight and the orthogonal walks, and since it is easier to code and does not encounter any problem with degenerate cases. The orthogonal walk can also be considered when an expensive arithmetic is used or when a large number of simplices must be traversed.

Open questions remain about the stochastic visibility walk. We showed that it always terminates, but it can have an exponential complexity on cases that are very pathologic, both in the choice of the triangulation and in the choice of the query point. It might be possible to get results under some hypotheses on the triangulation and on the query point: Is the expected complexity in the case of a Delaunay triangulation of n random points in dimension d equal to $\sqrt[d]{n}$? Would it be possible to get an amortized complexity for the successive locations of n points incrementally inserted into a Delaunay triangulation?

References

[1] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 11–18, 2000.

[2] P. Bose and L. Devroye. Intersections with random geometric objects. Technical report, School of Computer Science, McGill University, 1995. Manuscript.

[3] L. De Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. On sorting triangles in a Delaunay tessellation. *Algorithmica*, 6:522–532, 1991.

[4] Olivier Devillers. Improved incremental randomized Delaunay triangulation. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 106–115, 1998.

[5] Luc Devroye, Ernst Peter Mücke, and Binhai Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[6] H. Edelsbrunner. An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10(3):251–260, 1990.

[7] Paul-Louis George and Houman Borouchaki. *Triangulation de Delaunay et maillage. Applications aux éléments finis.* Hermes, Paris, France, 1997.

[8] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[9] C. Lemaire. *Triangulation de Delaunay et arbres multidimensionnels.* Thèse de doctorat en sciences, École des Mines de St-Etienne, France, 1997.

[10] Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1996.

[11] F. P. Preparata. Planar point location revisited. *Internat. J. Found. Comput. Sci.*, 1(1):71–86, 1990.

Appendix: pseudo-code

Algorithm *2D Straight Walk(q,p)*

```
// traverses the triangulation T, following the line segment from q to p.
// t = qrl is a triangle of T.
if orientation(rqp)<0 while orientation(lqp)<0 {
      r=l;
      t=neighbor(t through ql);
      l=vertex of t, l≠q, l≠r; }
else do {
      l=r;
      t=neighbor(t through qr);
      r=vertex of t, r≠q, r≠l;
      } while orientation(rqp)<0;
// end of initialization step
// now qp has r on its right and l on its left.
while orientation(prl)<0 {
   t = neighbor(t through rl);
   s = vertex of t, s≠r, s≠l;
   if orientation(sqp)<0 r=s; else l=s; }
// t contains p.
```

Algorithm *3D Straight Walk(q,p)*

```
// traverses the triangulation T, following the line segment from q to p
// t = uvwq is a tetrahedron of T.
if orientation(vuqp)>0 while orientation(wuqp)>0 {
      v=w;
      t=neighbor(t through quw);
      w=vertex of t, w≠u, w≠v, w≠q; }
else do {
      w=v;
      t=neighbor(t through quv);
      v=vertex of t, v≠u, v≠w, v≠q;
      } while orientation(vuqp)<0;
// now v and w lie on opposite sides of plane uqp,
// vuqp is positively oriented and wuqp negatively.
while orientation(vwqp)>0 {
   t = neighbor(t through qvw);
   s = vertex of t, s≠v, s≠w, s≠q;
   if orientation(suqp)>0 v=s; else w=s; }
u = vertex of t, s≠v, s≠w, s≠q;
// end of initialization step,
```

```
// qp intersects triangle uvw,
// wvqp, vuqp and uwqp are positively oriented.
while orientation(uwvp)>0 {
   t = neighbor(t through uvw);
   s = vertex of t, s≠u, s≠v, s≠w;
   if orientation(usqp)>0 // qp does not intersect triangle usw,
      if orientation(vsqp)>0 // qp intersects triangle vsw,
         u=s;
      else // qp intersects triangle usv,
         w=s;
   else // qp does not intersect triangle usv,
      if orientation(wsqp)>0 // qp intersects triangle usw,
         v=s;
      else // qp intersects triangle vsw,
         u=s;
// t contains p.
```

Algorithm *2D Orthogonal Walk(q,p)*

```
// traverses the triangulation T, using the orthogonal walk from q to p,
// t = qrl is a triangle of T.  wlog, we assume p is above and to the right of
q.
α=point(xp,yq);
if r below q while l below q {
      r=l; t=neigbor(t through ql); l=vertex of t≠qr;}
else do {
      l=r; t=neigbor(t through qr); r=vertex of t≠ql;
   } while r above q;
// q has r below and l above.
while ((r and l at left of α) or orientation(αrl)<0){
      t = neighbor(t through rl);
      s = vertex of t≠rl;
      if s above q l=s; else r=s; }
// α inside t
l = vertex of t≠rl;
r = vertex of t≠rl;
// p has r at right and l at left.
while ((r and l below p) or orientation(prl)<0) {
      t = neighbor(t through rl);
      s = vertex of t≠rl;
      if s at left of p l=s; else r=s;}
// t contains p.
```

Algorithm *Remembering Stochastic Walk(q,p)*

```
// traverses the triangulation 𝒯, using the remembering stochastic walk
// from q to p.  t = qrl is a triangle of 𝒯.
previous=t; end=false;
while (not end) {
   e = random edge of t;
   if (p not neighbor of previous through e) and (p on the other side of e)
            {previous=t;t=neighbor(t through e);}
   else {
      e = next edge of t;
      if (p not neighbor of previous through e) and (p on the other side of e)
            {previous=t;t=neighbor(t through e);}
      else {
         e = next edge of t;
         if (p not neighbor of previous through e) and (p on the other side of
e)
            {previous=t;t=neighbor(t through e);}
         else end=true;
      }
   }
}
// t contains p.
```

run times on 5 ≠ inputs	100.000 points						1.000.000 points					
	Walk			Hierarchy			Walk			Hierarchy		
	♯Δ per point	♯orient per point	μs	♯Δ per point	♯orient per point	μs	♯Δ per point	♯orient per point	μs	♯Δ per point	♯orient per point	μs
Stochastic 2D	73	131	64	23	49	40	192	334	142	28	60	47
	69	126	62	22	46	40	244	425	171	29	62	47
	68	124	62	23	49	40	264	459	184	28	60	47
	84	151	70	24	51	41	193	338	144	29	62	47
	72	130	64	23	49	41	230	398	164	28	59	47
Visibility 2D	80	138	65	21	41	38	285	488	183	29	55	47
	84	149	67	23	46	39	220	379	150	30	58	46
	63	112	57	22	43	39	180	310	129	29	57	46
	67	119	59	22	43	39	242	418	162	28	56	46
	92	159	70	23	45	39	194	334	135	28	54	46
Rem stoch 2D	61	111	60	23	49	41	235	409	176	29	62	49
	84	151	71	24	50	41	196	341	153	28	60	49
	72	131	68	22	48	41	213	371	164	30	62	49
	60	110	60	23	49	42	221	384	168	28	60	48
	69	124	64	23	49	41	232	404	176	28	60	48
Rem visib 2D	86	116	65	23	35	39	257	340	163	28	42	45
	70	95	59	22	34	38	254	337	162	28	42	45
	70	95	59	23	34	39	227	299	148	28	42	45
	96	128	71	22	34	38	177	233	123	27	41	45
	61	85	55	23	34	38	204	268	136	28	42	45
Straight 2D	94	188	80	20	43	43	168	337	142	23	51	49
	75	151	69	21	44	43	189	380	155	22	49	49
	57	116	60	20	42	43	224	449	178	24	51	49
	71	146	68	18	40	42	200	402	163	25	54	51
	59	121	62	19	40	42	180	363	151	23	51	50
Orthogonal 2D	84	5	61	24	8	41	223	5	133	31	11	49
	95	6	65	27	8	42	269	5	161	31	10	49
	76	5	60	28	8	43	288	5	167	34	11	51
	86	5	62	26	9	42	333	5	180	33	10	50
	78	5	60	28	8	43	300	5	165	33	10	50
Orthog visib 2D	116	97	76	36	36	45	301	245	170	44	49	54
	101	83	69	39	39	46	263	196	150	43	42	52
	95	71	66	35	35	52	261	191	149	40	42	52
	107	91	71	35	37	44	416	306	213	42	45	52
	96	72	66	37	36	47	327	247	176	41	44	53
Visibility 3D	171	353	175	29	73	64	400	819	467	34	84	80
	182	380	187	28	71	64	369	757	428	36	87	81
	186	382	186	31	76	66	324	665	383	37	88	80
	181	380	185	29	72	64	375	765	436	38	91	81
	197	406	197	32	78	67	326	670	383	34	82	80
Stochastic 3D	158	308	174	29	69	67	354	679	424	34	79	83
	167	326	181	29	69	64	343	657	410	33	77	80
	168	327	182	30	70	65	315	604	380	33	77	80
	167	326	190	28	66	64	340	655	409	37	85	83
	172	334	186	29	68	64	318	610	383	35	81	80
Rem visib 3D	171	280	167	30	59	65	400	646	444	36	69	81
	182	299	179	30	58	66	369	598	415	34	65	79
	186	301	176	31	61	65	324	524	365	35	67	83
	181	299	176	30	57	64	375	606	422	34	65	80
	197	320	188	29	58	65	326	528	367	36	68	81
Rem stoch 3D	158	247	168	29	56	64	354	547	412	34	63	78
	167	262	177	30	57	64	343	529	400	35	66	81
	168	263	177	30	57	64	315	486	372	36	66	82
	167	262	176	29	55	64	340	527	402	35	65	81
	172	269	180	29	55	63	318	490	371	34	64	81
Straight 3D	146	434	192	24	74	70	343	1022	477	31	91	85
	155	462	203	24	71	68	319	950	453	32	96	86
	159	472	206	26	77	70	278	827	397	30	89	86
	155	460	202	25	75	69	323	963	457	30	89	87
	168	501	216	25	74	71	280	834	402	30	90	85
Orthogonal 3D	214	8	204	41	20	85	522	8	538	47	21	106
	192	15	205	44	20	86	487	8	496	48	21	108
	198	8	217	40	20	84	326	19	398	49	21	108
	182	8	191	41	20	85	340	15	406	48	21	110
	199	11	212	41	20	85	410	8	421	48	21	111

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399